Predicting Play Calling Success in the NFL

In this notebook, I train various machine learning models to predict the success of a "run" or "pass" play call in the NFL. The notebook is accompanied by another Jupyter Notebook where I clean the NFL data for easy use by these algorithms. See that notebook if you want to know more about the original dataset and how to clean data using the Pandas library.

Before I begin my analysis, I find it appropriate to present my opinion on how analytics should be used in sports. While I think that analytics has a incredibly high use potential in sports, I do see its limitations. Analytics recommendations should always be taken with the knowledge of what assumptions the recommendation is making, the limitations of the recommendation system, and the knowledge that probabilities are just probabilities, not certainties. With that out of the way, let's dive into the analysis of this data set.

This notebook presents the prediction accuracy of four different machine learning algorithms as applied to the NFL Play-by-Play data set found here (https://www.kaggle.com/datasets/toddsteussie/nfl-play-statistics-dataset-2004-to-present). Three of the algorithms I use are tree-based machine learning algorithms (random forest with 2 different representations of the data and one boosted tree algorithm) and one of them is a neural network.

My goal with this algorithm is to be able to recommend whether an offensive coordinator should call a rushing or passing play based on statistical analysis of past data. What my algorithm does is use pre-snap data to predict whether a rushing or passing play will be a "success", "failure", or "neutral". The definition of what a success is is found below.

Rules For Success

Below are a description of the rules used by the add_success function to evaluate whether each play was a success or not. I chose these rules based on what I feel should be considered a successful football play. The rules are different depending on what down it is. In the data set, successes are represented by 1, failures by -1, and neutral plays by 0.

Any down:

- if you get a first down or score, that's a success

First down:

- success = gaining (3/10) or greater of yards to the sticks
- neutral = between 1 yard and (3/10) of yards to the sticks
- fail = 0 yards or negative yards

Second down:

- success = gaining (1/2) or greater of yards to the sticks
- neutral = between 1 yard and (1/2) of yards to the sticks
- fail = 0 or negative yards

Third down:

- success = first down
- neutral = nothing
- fail = not first down

Fourth down:

- success = first down
- neutral = nothing
- fail = not first down

Results

I find it useful to present the results of the analysis first before presenting the analysis itself.

In my opinion, the best metric to decide which algorithm is "best" in this context, is the algorithms recall precentage for successful plays. This is the percentage of times that my algorithm predicts a successful play, where the play is, in fact, successful. In other words, if my algorithm predicts you will have a successful play-call, then I have a 57% chance of being right. This may sound low, but just consider the following fact: less than 49% of total play calls end up in a success. We can reasonably assume that every play call made by an offensive coordinator/head coach is predicted to be a success, therefore my best algorithm beats the average head coach by roughly 8%!!

A table summarizing each of the algorithms results is show below. Remember that (at least in my opinion) the precision on successful plays is the most important metric.

Random Forest with unnormalized data

In [43]:
```
print(skl.metrics.classification_report(y_test, preds, target_names=target_name
```

```
                precision    recall  f1-score   support

    failed play       0.54      0.48      0.51     78361
   neutral play       0.44      0.20      0.28     26532
successful play       0.55      0.67      0.61    102256

       accuracy                           0.54    207149
      macro avg       0.51      0.45      0.46    207149
   weighted avg       0.53      0.54      0.53    207149
```

Random Forest with noramlized data

In [44]:
```
print(skl.metrics.classification_report(y_trans_test, preds_trans, target_names
```

```
                precision    recall  f1-score   support

    failed play       0.55      0.45      0.50     78361
   neutral play       0.42      0.24      0.31     26532
successful play       0.55      0.69      0.62    102256

       accuracy                           0.54    207149
      macro avg       0.51      0.46      0.47    207149
   weighted avg       0.54      0.54      0.53    207149
```

XGBoost Boosted Trees

In [50]:
```
print(skl.metrics.classification_report(y_test, xgb_preds, target_names=target_
```

```
                precision    recall  f1-score   support

    failed play       0.53      0.52      0.53     78361
   neutral play       0.38      0.34      0.36     26532
successful play       0.58      0.61      0.60    102256

       accuracy                           0.54    207149
      macro avg       0.50      0.49      0.49    207149
   weighted avg       0.54      0.54      0.54    207149
```

TensorFlow Neural Network

In [51]:
```
print(skl.metrics.classification_report(y_test, preds_nn, target_names=target_r
```

```
                precision    recall  f1-score   support

    failed play       0.62      0.39      0.48     78361
   neutral play       0.50      0.24      0.33     26532
successful play       0.56      0.80      0.66    102256

       accuracy                           0.57    207149
      macro avg       0.56      0.48      0.49    207149
   weighted avg       0.58      0.57      0.55    207149
```

Analysis

Presented below is the full code used to train each of the algorithms on our data set.

In [2]:
```
%%capture
import numpy as np
import pandas as pd

import sklearn as skl
from sklearn import preprocessing
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.compose import make_column_transformer
```

```
from sklearn.preprocessing import StandardScaler

import tensorflow as tf
from tensorflow import keras
import keras_tuner as kt

import matplotlib.pyplot as plt
import seaborn as sns
```

```
/Users/Michaelray/opt/anaconda3/envs/local/lib/python3.7/site-packages/tensorf
low/python/framework/dtypes.py:516: FutureWarning: Passing (type, 1) or '1typ
e' as a synonym of type is deprecated; in a future version of numpy, it will b
e understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
/Users/Michaelray/opt/anaconda3/envs/local/lib/python3.7/site-packages/tensorf
low/python/framework/dtypes.py:517: FutureWarning: Passing (type, 1) or '1typ
e' as a synonym of type is deprecated; in a future version of numpy, it will b
e understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
/Users/Michaelray/opt/anaconda3/envs/local/lib/python3.7/site-packages/tensorf
low/python/framework/dtypes.py:518: FutureWarning: Passing (type, 1) or '1typ
e' as a synonym of type is deprecated; in a future version of numpy, it will b
e understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
/Users/Michaelray/opt/anaconda3/envs/local/lib/python3.7/site-packages/tensorf
low/python/framework/dtypes.py:519: FutureWarning: Passing (type, 1) or '1typ
e' as a synonym of type is deprecated; in a future version of numpy, it will b
e understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
/Users/Michaelray/opt/anaconda3/envs/local/lib/python3.7/site-packages/tensorf
low/python/framework/dtypes.py:520: FutureWarning: Passing (type, 1) or '1typ
e' as a synonym of type is deprecated; in a future version of numpy, it will b
e understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
/Users/Michaelray/opt/anaconda3/envs/local/lib/python3.7/site-packages/tensorf
low/python/framework/dtypes.py:525: FutureWarning: Passing (type, 1) or '1typ
e' as a synonym of type is deprecated; in a future version of numpy, it will b
e understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
/Users/Michaelray/opt/anaconda3/envs/local/lib/python3.7/site-packages/tensorb
oard/compat/tensorflow_stub/dtypes.py:541: FutureWarning: Passing (type, 1) or
'1type' as a synonym of type is deprecated; in a future version of numpy, it w
ill be understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
/Users/Michaelray/opt/anaconda3/envs/local/lib/python3.7/site-packages/tensorb
oard/compat/tensorflow_stub/dtypes.py:542: FutureWarning: Passing (type, 1) or
'1type' as a synonym of type is deprecated; in a future version of numpy, it w
ill be understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
/Users/Michaelray/opt/anaconda3/envs/local/lib/python3.7/site-packages/tensorb
oard/compat/tensorflow_stub/dtypes.py:543: FutureWarning: Passing (type, 1) or
'1type' as a synonym of type is deprecated; in a future version of numpy, it w
ill be understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
/Users/Michaelray/opt/anaconda3/envs/local/lib/python3.7/site-packages/tensorb
oard/compat/tensorflow_stub/dtypes.py:544: FutureWarning: Passing (type, 1) or
'1type' as a synonym of type is deprecated; in a future version of numpy, it w
ill be understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
/Users/Michaelray/opt/anaconda3/envs/local/lib/python3.7/site-packages/tensorb
oard/compat/tensorflow_stub/dtypes.py:545: FutureWarning: Passing (type, 1) or
'1type' as a synonym of type is deprecated; in a future version of numpy, it w
ill be understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
/Users/Michaelray/opt/anaconda3/envs/local/lib/python3.7/site-packages/tensorb
oard/compat/tensorflow_stub/dtypes.py:550: FutureWarning: Passing (type, 1) or
'1type' as a synonym of type is deprecated; in a future version of numpy, it w
ill be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
```

In [3]:
```
cd archive\ (3)
```

```
/Users/Michaelray/Documents/NFL_project/archive (3)
```

In [4]:
```
md = pd.read_csv('model_data.csv')
md = md.drop(columns=['Unnamed: 0'])
```

In [5]:
```
X, y = md.drop(columns=['success']), md['success']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, rando
```

Random Forest with unnormalized data

In [41]:
```
%%capture
rf = RandomForestClassifier()
rf.fit(X_train, y_train)
```

In [42]:
```
target_names = ['failed play', 'neutral play', 'successful play']
preds = rf.predict(X_test)
print(skl.metrics.classification_report(y_test, preds, target_names=target_name
```

```
                precision    recall  f1-score   support

    failed play       0.54      0.48      0.51     78361
   neutral play       0.44      0.20      0.28     26532
successful play       0.55      0.67      0.61    102256

       accuracy                           0.54    207149
      macro avg       0.51      0.45      0.46    207149
   weighted avg       0.53      0.54      0.53    207149
```

Random Forest with normalized data

In [8]:
```
cols = ['playId', 'gameId', 'playSequence', 'quarter', 'playNumberByTeam',
                'gameClock', 'down', 'distance', 'distanceToGoalPre', 'evPre',
              'homeScorePre', 'visitingScorePre', 'huddle',

              'Bears home', 'Bengals home', 'Bills home',
              'Broncos home', 'Browns home', 'Buccaneers home', 'Cardinals home
              'Chiefs home', 'Colts home', 'Cowboys home', 'Dolphins home', 'Ea
              'Falcons home', 'Giants home', 'Jaguars home', 'Jets home', 'Lion
              'Niners home', 'Packers home', 'Panthers home', 'Patriots home',
              'Rams home', 'Ravens home', 'Saints home', 'Seahawks home', 'Stee
              'Titans home', 'Vikings home', 'Washington home', 'Bears away', '
              'Broncos away', 'Browns away', 'Buccaneers away', 'Cardinals away
              'Chiefs away', 'Colts away', 'Cowboys away', 'Dolphins away', 'Ea
              'Giants away', 'Jaguars away', 'Jets away', 'Lions away', 'Niners
              'Panthers away', 'Patriots away', 'Raiders away', 'Rams away', 'P
              'Seahawks away', 'Steelers away', 'Texans away', 'Titans away', '


                'homeTeamPossession', 'pass', 'rush', 'success']

normalizables = ['playId', 'gameId', 'playSequence', 'quarter', 'playNumberByTe
                'gameClock', 'down', 'distance', 'distanceToGoalPre', 'evPre',
              'homeScorePre', 'visitingScorePre']
```

In [9]:
```
#Normalize all numerical variables
scaler = StandardScaler()
ct = make_column_transformer((scaler, normalizables), remainder='passthrough')
md_trans = ct.fit_transform(md)
md_trans = pd.DataFrame(md_trans, columns=cols)
```

In [10]:
```
#Make new test, train data
X_trans, y_trans = md_trans.drop(columns=['success']), md_trans['success']
X_trans_train, X_trans_test, y_trans_train, y_trans_test = train_test_split(X_t
```

In [11]:
```
%%capture
rf2 = RandomForestClassifier()
rf2.fit(X_trans_train, y_trans_train)
preds_trans = rf2.predict(X_trans_test)
```

In [12]:
```
print(skl.metrics.classification_report(y_trans_test, preds_trans, target_names
```

```
                precision    recall  f1-score   support

    failed play       0.55      0.45      0.50     78361
   neutral play       0.42      0.24      0.31     26532
successful play       0.55      0.69      0.62    102256

       accuracy                           0.54    207149
      macro avg       0.51      0.46      0.47    207149
   weighted avg       0.54      0.54      0.53    207149
```

XG Boost

In [46]:
```
import xgboost as xgb
```

In [47]:
```
%%capture
#Higher lambda and higher gamma makes a more conservative tree
#Higher max_deptch means more likelihood of overfitting
xgb_classifier = xgb.XGBClassifier(
    n_estimators=100,
    reg_lambda=1,
    gamma=0,
    max_depth=3
)
xgb_classifier.fit(X_trans_train, y_train_nn)
xgb_preds = np.argmax(xgb_classifier.predict(X_trans_test), axis=1)
```

In [49]:
```
target_names = ['failed play', 'neutral play', 'successful play']
print(skl.metrics.classification_report(y_test, xgb_preds, target_names=target_
```

```
                precision    recall  f1-score   support

    failed play       0.53      0.52      0.53     78361
   neutral play       0.38      0.34      0.36     26532
successful play       0.58      0.61      0.60    102256

       accuracy                           0.54    207149
      macro avg       0.50      0.49      0.49    207149
   weighted avg       0.54      0.54      0.54    207149
```

Neural Network

In [32]:
```
#To use loss function in NN, we need to one-hot encode the target variables
y_train_nn = pd.get_dummies(y_train)
#y_train_nn = y_train_nn.rename(columns = {-1 : 'failedPlay', 0 : 'neutralPlay
y_test_nn = pd.get_dummies(y_test)
#y_test_nn = y_test_nn.rename(columns = {-1 : 'failedPlay', 0 : 'neutralPlay',
```

In [33]:
```
# Tranform target data from -1, 0, 1 to 0, 1, 2
X, y = md.drop(columns=['success']), md['success']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, rando
y_train.loc[y_train==1] = 2
y_train.loc[y_train==0] = 1
y_train.loc[y_train==(-1)] = 0
y_test.loc[y_test==1] = 2
y_test.loc[y_test==0] = 1
y_test.loc[y_test==(-1)] = 0

y_train_nn = pd.get_dummies(y_train)
y_test_nn = pd.get_dummies(y_test)
```

In [34]:
```
%%capture
model = tf.keras.models.Sequential()

model.add(tf.keras.layers.InputLayer(input_shape = (X_train.shape[1], )))
model.add(tf.keras.layers.Dense(18, activation='relu'))
model.add(tf.keras.layers.Dense(9, activation='relu'))
model.add(tf.keras.layers.Dense(3, activation = 'softmax'))

model.compile(optimizer = 'adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=F
              metrics = ['accuracy'])

model.fit(X_trans_train, y_train, epochs = 10)
```

```
WARNING:tensorflow:From /Users/Michaelray/opt/anaconda3/envs/local/lib/python
3.7/site-packages/tensorflow/python/ops/init_ops.py:1251: calling VarianceScal
ing.__init__ (from tensorflow.python.ops.init_ops) with dtype is deprecated an
d will be removed in a future version.
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to the
constructor
2022-05-15 16:04:11.840075: I tensorflow/core/platform/cpu_feature_guard.cc:14
2] Your CPU supports instructions that this TensorFlow binary was not compiled
to use: AVX2 FMA
```

In [35]:
```
preds_nn = np.argmax(model.predict(X_trans_test), axis=1)
target_names = ['failed play', 'neutral play', 'successful play']
print(skl.metrics.classification_report(y_test, preds_nn, target_names=target_r
```

	precision	recall	f1-score	support
failed play	0.62	0.39	0.48	78361
neutral play	0.50	0.24	0.33	26532
successful play	0.56	0.80	0.66	102256
accuracy			0.57	207149
macro avg	0.56	0.48	0.49	207149
weighted avg	0.58	0.57	0.55	207149